SRM V2.2 Specification

Timur Perelmutov, FNAL on behalf of WLCG Data Management Coordination Group and Storage Resource Manager Collaboration.

Wednesday, May 11, 2006

Draft Version 3

Abstract

In order to satisfy the requirements of the LHC Experiments of the Grid Storage Interfaces we introduce modifications described bellow to SRM V2.1.1 specification, and propose to call the new interface SRM V2.2.

SRM Version 2.2 Specification extends and supersedes the SRM V2.1 and V2.1.1 specification.

SRM V2.1 style definitions are used in this document. The yellow background highlights the new data types and functions and the new members of the existing data structures.

Table of Context

Storage Classes

Storage Class will be a property of the file that describes the minimum number of copies that should be stored on each type of the media supported by the system. Currently we limit the media types to just Disk and Tape.

Motivation

Provide the motivation here.

TStorageClass enumeration type

A new enumeration type TStorageClass will have the following legal values:

```
enum            TStorageClass       { Tape0Disk1,Tape0DiskN,
                                      Tape1Disk0,Tape1Disk1,
                                      Tape1DiskN,TapeNDisk0,
                                      TapeNDisk1,TapeNDiskN}
```

Alternatively the TStorageClass will have the following definition

```
enum            TStorageClass       {Replica,
                                     Output,
                                     Custodial }
```

Description

Each value in the enumeration is a string that is composed of the word “Tape”, followed by the character “0”, “1” or “N”, followed by the word “Disk” followed again by the character “0”, “1” or “N”. The character following the word “Tape” or the word “Disk” specifies the minimum number of copies of files stored on tape or disk media , “0” means no copies are required, “1” means at least one copy is required and “N” means 2 or more copies are required to exist.

Alternative Description

Replica Storage Class means that the data is stored on the media that has high probability of loss, for LHC usage it might be considered to be stored on disk only.

Output Storage Class means that the data is much less likely to be lost compared with Replica Class , but there is still some chance of it, so the data stored in this Storage Class should be reproducible. For LHC usage it can be assumed to be stored on raid array type of disks.

Custodial Storage Class means that the probability of the data loss is extremely low and it is appropriate for the storage of the unique unrecoverable data. This Storage Class can be assumed to correspond to taped backed up storage

Usage

The type TStorageClass will be used to describe Storage Class assigned to the files in the storage system, at the moments when the files are written into the system.

Also the TStorageClass will now be a property of space reservations allocated through the Space Management functions of dCache. Once the Storage Class is

assigned to a Space, the files put in the reserved space will automatically be assigned the Storage Class of the Space if TStorageClass type parameter is omitted in srmPrepareToPut or srmCopy function invocation.

Information about the storage class assigned to the file will be discoverable thought the TMetaDataPathDetail structure returned for each file by the srmLs function, which will now have a new parameter called assignedStorageClass.

When a file is only written to storage system, it usually has only one copy on one type of media, and then storage system makes additional copies according to the Storage Class assigned to the file. While this process takes place, users want to know the actual number of copies on each type of media. In order to accommodate this desire we introduce a new member of the TMetaDataPathDetail structure returned by the srmLs function, namely currentStorageClass.

CurrentStorageClass field of TMetaDataPathDetail will always reflect the current state of the file, regardless of the assigned Storage Class. If the file is present on disk, the currentStorageClass will be Tape1 Disk1 even if assigned Storage Class is Tape1Disk0.

TMetaDataSpace and TMetaDataPathDetail modifications

Therefore the new definition of the TMetaDataSpace and TMetaDataPathDetail is as follows:

```
typedef     struct {TSpaceType          type,
                    TStorageClass       storageClass,
                    TSpaceToken         spaceToken,
                    Boolean             isValid,
                    TUserID             owner,
                    TSizeInBytes        totalSize,      // best effort
                    TSizeInBytes        guaranteedSize,
                    TSizeInBytes        unusedSize,
                    TLifeTimeInSeconds  lifetimeAssigned,
                    TLifeTimeInSeconds  lifetimeLeft
                } TMetaDataSpace

typedef     struct {string              path,   // both dir and file
                    TReturnStatus       status,
                    TSizeInBytes        size,   // 0 if dir
                    TOwnerPermission    ownerPermission,
                    TUserPermission[]   userPermission,
                    TGroupPermission[]  groupPermission,
                    TOtherPermission    otherPermission
                    TGMTTime            createdAtTime,
                    TGMTTime            lastModificationTime,
                    TUserID             owner,
```

```
        TFileStorageType        fileStorageType,
        TFileType               type,    // Directory or File
        TStorageClass           assignedStorageClass,
        TStorageClass           currentStorageClass,
        TLifeTimeInSeconds      lifetimeAssigned,
        TLifeTimeInSeconds      lifetimeLeft,
        TCheckSumType           checkSumType,
        TCheckSumValue          checkSumValue,
        TSURL                   originalSURL,   // if path is a file
        TMetaDataPathDetail[]   subPath         // optional recursive
} TMetaDataPathDetail
```

srmChangeStorageClass function

LCG Experiments expressed the desire to be able to change the storage class of the existing data in the storage system. In order to do so we introduce a new function **srmChangeStorageClass.**

The function will have the following input and output parameters.

srmChangeStorageClass

```
In:     TUserID                 authorizationID,
        TChangeStorageType []   arrayOfChangeRequest,

Out:    TReturnStatus           returnStatus,
        TSURLReturnStatus[]     arrayOfFileStatus
```

Where TChangeStorageType is defined as

```
typedef     struct {
                TSURLInfo           surlInfo,
                TStorageClass       storageClass
            } TChangeStorageType
```

Description

srmChangeStorageClass will take an array of file path structures describing the files that client wants to assign a new Storage Class . The assignedStorageClass parameter describes what is the desired Storage Class. The function will return a returnStatus, telling if the execution took place and if there were some conditions preventing the execution such as lack of authorization, etc. The arrayOfFileStatus returned to the client will describe the success of the request (or lack of it) for each individual file in the request.

TAccessPattern type for description of the expected TURL utilization

In order to better accommodate the transfer request initiated by srmPrepareToGet and srmPrepareToPut the following new data types are introduced:

```
enum        TTransferPattern     { "TransferLimited",
                                   "ApplicationLimited" }


enum        TConnectionType      { "WAN",
                                   "LAN" }
```

<!-- Timur: The earlier proposed terms "Sequential" and "Random" did not reflect the qualities of the transfer we were interested in. "Random" access transfer still can go at a very high transfer rate and "Sequential" one can be very slow. So the new terms are offered, and if collaboration can propose a better terminology, it will be gladly accepted. -->

Description

TTransferPattern type TConnectionType parameters will be passed as inputs to the srmPrepareToGet or srmPrepareToPut function. It will indicate how the transfer URL produced by srm is going to be used. If the parameter value is "ApplicationLimited", the system might expect that application will perform some processing of the partially read data, followed by more partial reads and a frequent use of the protocol specific "seek" operation. If the value is "TansferLimited" the file will be read at the highest speed allowed by the connection between server and a client. TConnectionType will indicate if the client is connected though a local or wide area network. The srm will then have a chance to optimize the transfer parameters to achieve maximim thoughput for the configuration.

<!-- Timur : There were other parameters discussed , like access speed, file system parametes tcp/ip transfer protocol properties, but I did not find these as useful, and I do not remember that there was a consensus about what to include-->

srmPrepareToGet, srmPrepareToPut and srmCopy Modifications

In order to allow the specification of the StorageClass of the files transferred into the system and allow to specify the access patters for the transfers between the Storage System and the client srmPrepareToGet, srmPrepareToPut and srmCopy input arguments need to be changed. While the signatures of the data transfer functions srmPrepareToGet, srmPrepareToPut and srmCopy remain the same, the following data types used as types of input parameters for the functions mentioned here are changed and now look as following:

```
typedef     struct {TSURLInfo                fromSURLInfo,
                    TLifeTimeInSeconds       lifetime, // pin time
                    TFileStorageType         fileStorageType,
```

```
                    TSpaceToken             spaceToken,
                    TDirOption              dirOption ,
                    TTransferPattern        transferPattern,
                    TConnectionType         connectionType
                } TGetFileRequest

typedef     struct {TSURLInfo               toSURLInfo,   // local to SRM
                    TLifeTimeInSeconds      lifetime,     // pin time
                    TFileStorageType        fileStorageType,
                    TSpaceToken             spaceToken,
                    TSizeInBytes            knownSizeOfThisFile,
                    TTransferPattern        transferPattern,
                    TConnectionType         connectionType,
                    TStorageClass           storageClass
                } TPutFileRequest

typedef     struct {TSURLInfo               fromSURLInfo,
                    TSURLInfo               toSURLInfo,
                    TLifeTimeInSeconds      lifetime, // pin time
                    TFileStorageType        fileStorageType,
                    TSpaceToken             spaceToken,
                    TOverwriteMode          overwriteMode,
                    TDirOption              dirOption,
                    TTransferPattern        transferPattern,
                    TConnectionType         connectionType,
                    TStorageClass           storageClass
} TCopyFileRequest
```

srmReserveSpace Changes

It is expected that in a distributed Mass Storage System space reservation operation will take a significant amount of time to complete. We feel that making the srmReserveSpace operation asynchronous we will make the servers capable of supporting larger number of concurrent clients and make clients capable of submitting multiple reservation requests and starting a transfer into the successful ones without using multiple threads and multiple connections to the server.

To support asynchronous execution of the srmReserveSpace function we add the following definition of TReserveSpaceRequest type:

```
typedef     struct {TSpaceType              typeOfReservedSpace,
                    TSizeInBytes            sizeOfTotalReservedSpace, // best
                effort
                    TSizeInBytes            sizeOfGuaranteedReservedSpace,
                    TLifeTimeInSeconds      lifetimeOfReservedSpace,
```

	TSpaceToken,	referenceHandleOfReservedSpace,
	TStorageClass	defaultStorageClass,
	TReturnStatus	status

} **TReserveSpaceRequestStatus**

In order to allow the asynchronous invocation and the specification of the default StorageClass for the files that will be transferred into the reserved the **srmReserveSpace** will now be defined as follows:

srmReserveSpace

In:	TUserID	authorizationID,
	TSpaceType	typeOfSpace,
	String	userSpaceTokenDescription,
	TSizeInBytes	sizeOfTotalSpaceDesired,
	TSizeInBytes	sizeOfGuaranteedSpaceDesired,
	TLifeTimeInSeconds	lifetimeOfSpaceToReserve,
	TStorageSystemInfo	storageSystemInfo
	TStorageClass	storageClass
Out:	TReserveSpaceRequestStatus	reserveSpaceStatus,
	TRequestToken	reserveSpaceRequestToken,
	TLifeTimeInSeconds	expectedTimeToCompletion,
	TReturnStatus	returnStatus

We also add the following new function:

srmStatusOfReserveSpaceRequest

In:	TRequestToken	reserveSpaceRequestToken,
	TUserID	authorizationID
Out:	TReturnStatus	returnStatus,
	TLifeTimeInSeconds	expectedTimeToCompletion,
	TReserveSpaceRequestStatus	reserveSpaceStatus

The execution of the srmReserveSpace can now proceed in two different ways:
1. Synchronous execution – space is available immediately, the **srmReserveSpace** returns SRM_SPACE_AVAILABLE as returnStatus and fills in the values of the reserveSpaceStatus structure.
2. Asyncronous execution – if it will take long time to execute the request, the srmReserveSpace returns SRM_REQUEST_QUEUED or SRM_REQUEST_INPROGRESS as a value of returnStatus and gives a client a valid reserveSpaceRequestToken. The token than can be utilized in **srmStatusOfReserveSpaceRequest** to get the current status of the reserve space request. Once request is completed, the **srmStatusOfReserveSpaceRequest** will returns SRM_SPACE_AVAILABLE as returnStatus and fills in the values of the reserveSpaceStatus structure.

At any point **srmReserveSpace and srmStatusOfReserveSpaceRequest** can return one of the errors as the values of the returnStatus. At this point the execution of request is completed (failed) and further executions of **srmStatusOfReserveSpaceRequest** with the same token will return exactly the same returnStatus.

srmAbortRequest abort request can be used to terminate the reserve space request prematurely.

srmBringOnline and getOnlineFileStatus functions

In order to satisfy the desire of experiments to be able to make certain data readily available for future transfers, without burdening the system with calculation of the Transfer URLs and other operations associated with srmPrepareToGet function, we introduce the new data type **TOnlineStateClaim** and a new function **srmBringOnline,** which in hieratical mass storage system is expected to stage date to the top of hierarchy (this operations is also known as "Staging") and making sure that the data stays there for a certain period of time (which is also known as "Pinning"). The execution of the function results in the "Bring Online File Status" created on the server, which has its own lifetime. The file is expected to remain online for as long as there are active "Bring Online File Status".

typedef	struct {TSURL	SURLInfo,
	TLifeTimeInSeconds	remainingTimeInOnlineState,
	TReturnStatus	status
	} TBringOnlineFileStatus	

srmBringOnline

In:	TUserID	authorizationID,
	string[]	arrayOfTransferProtocols,
	TSURLInfo[]	arrayOfFilePaths,
	TLifeTimeInSeconds	lifetimeOfOnlineState,
	TTransferPattern	transferPattern,
	TConnectionType	connectionType,
	TSpaceToken	spaceToken,
Out:	TReturnStatus	returnStatus,
	TRequestToken	requestToken,
	TBringOnlineFileStatus []	arrayOfFileStatus

If the lifetimeOfOnlineState is not specified, the system wide or VO wide defaults should be used. The requestToken can be used to terminate the claim(s) on file(s)'s online state

prematurely by the AbortRequest, AbortFiles. Lifetime of the claim can be extended using ExtendFileLifeTime function.

The user has ability to discover the remaining lifetime of the online state claims on basis of the list of the online state claim token and optrionaly the names of the files using the **getOnlineFileStatus**.

getOnlineFileStatus

In:	TUserID	authorizationID,
	TRequestToken	requestToken,
	TSURLInfo[]	arrayOfFilePaths,
Out:	TReturnStatus	returnStatus,
	TBringOnlineFileStatus []	arrayOfFileStatus

Resources

[1] The Storage Resource Manager Interface Specification, Version 2.1.1, Junmin Gu, Alex Sim, Arie Shoshani and SRM Collaboration.
[2] SRM Storage and File Types V4, James Casey, Jean-Philippe Baud and WLCG Data Management Coordination Group
[3] Documents and notes from LCG Computing Service / Service Challenge 4 Workshop